ARHITECTURA CALCULATOARELOR 2003/2004
CURSUL 5

2.1.3 Instrucţiuni în cod maşină

Figura 2.2 prezintă câteva tipuri de instrucţiuni pe care o unitate centrală de prelucrare uzuală trebuie să le poate executa. Asemenea instrucţiuni sunt denumite **instrucţiuni în cod maşină (machine instructions).** Poate veţi fi surprinşi să aflaţi că lista instrucţiunilor în cod maşină este destul de scurtă. Unul dintre aspectele fascinante ale informaticii este acela că odată ce un calculator poate executa anumite operaţii elementare, simple dar bine alese, adăugarea de facilităţi suplimentare nu duce la creşterea capacităţilor teoretice ale calculatorului. Cu alte cuvinte, dincolo de un anumit punct, adăugarea de noi caracteristici poate duce la îmbunătăţirea comodităţii de utilizare şi a vitezei de lucru, dar nu creşte cu nimic abilităţile primare ale calculatorului. Atunci când ne referim la instrucţiunile cunoscute de un calculator, este util ca să observăm că ele pot fi clasificate în trei categorii:grupul instrucţiunilor de transfer, grupul instrucţiunilor aritmetico-logice şi grupul instrucţiunilor de control.

➢ Instrucţiuni de tranfer de date

Primul grup de instrcţiuni conţine instrucţiunile necesare pentru deplasarea datelor dintr-un loc în altul. Paşii 1, 2 şi 4 din figura 2.2 intră în această categorie. Ca şi în cazul memoriei principale, nu se obişnuieşte ca datele care sunt transferate într-o locaţie oarecare din calculator să fie şterse din locaţia iniţială. Din acest punct de vedere, termenul de transfer (tranfer) sau mutare (move), deşi utilizat frecvent, este inadecvat, mai degrabă fiind potriviţi termeni, cum ar fi copiere (copy) sau clonare (clone). Legat de terminologie, trebuie să menţionăm că atunci când ne referim la transferul datelor între CPU şi memoria principală a calculatorului se utilizează termeni speciali. Cererea de încărcare a unui registru de uz general nu conţinutul unei celule de memorie este desemnată ca o instrucţiune LOAD; invers, transferul conţinutului unui registru într-o celulă de memorie se face prin intermediul unei instrucţiuni STORE. În figura 2.2, paşii 1 şi 2 reprezintă instrucţiuni LOAD, iar pasul 4 este o instrucţiune STORE.

Un grup important de instrucţiuni din categoria celor de transfer de date este dedicat comenzilor pentru comunicaţia cu dispozitive din afara binomului CPU – memorie principală. Deoarece aceste instrucţiuni se ocupă de operaţiile de intrare/ieşire (input/output –I/O) din calculator, ele sunt denumite instrucţiuni de intrare/ieşire (I/O) şi uneori se consideră că ar constitui un grup distinct de instrucţiuni Pe de altă parte, subcapitolul 2.6 arată că adesea aceste operaţii de intrare/iesire sunt tratate utilizându-se aceleaşi instrucţiuni ca şi pentru transferul datelor între CPU şi memoria principală, aşa că plasarea lor într-o categorie distinctă nu este pe deplin justificată.

➢ Instrucţiuni aritmetice şi logice

Grupul instrucţiunilor aritmetice şi logice constă din instrucţiunile care indica unităţi de comandă să solicite unităţii aritmetico-logice efectuarea unei anumite operaţii. Pasul 3 din figura 2.2 face parte din acest grup de instrucţiuni. Aşa cum sugerează şi numele, unitatea

aritmetico-logică este capabilă să efectueze şi alte operaţii în afara operaţiilor aritmetice elementare. Unele dintre aceste operaţii sunt operaţiile logice AND, OR şi XOR, prezentate deja în capitolul 1 şi pe care le vom aprofunda mai târziu în acest capitol. Ele sunt utilizate adesea pentru manipularea individuală a biţilor dintr-un registru, fără modoficarea restului registrului. Un alt grup de operaţii care se pot efectua în cadrul unităţii aritmetico-logice permite deplasarea la dreapta sau la stânga a conţinutului regiştrilor. Aceste operaţii sunt denumite SHIFT sau ROTATE, după cum sunt trataţi biţii care ies din registru prin deplasarea conţinutului:biţii sunt pur şi simplu eliminaţi (SHIFT) sau sunt folosiţi pentru completarea golului care apare la celălalt capăt al registrului (ROTATE).

➢ Instrucţiuni de control

Grupul instrucţiunilor de control conţine acele instrucţiuni care nu manipulează date, ci dirijează modul de execuţie al programului. Pasul 5 din figura 2.2 face parte din această categorie, deşi constituie un caz elementar. Această categorie de instrucţiuni conţine multe dintre cele mai interesante instrucţiuni din limbajul calculatorului, cum ar fi familia de instrucţiuni de salt JUMP (sau BRANCH), care sunt utilizate pentru a face ca unitatea de comandă să execute altă instrucţiune decât cea care urmează. Există două variante de instrucţinui JUMP : salturi necondiţionate şi salturi condiţionate. Un exemplu din primul tip este “Salt la pasul 5”; un exemplu de salt condiţionat este următorul “Dacă valoarea obţinută este 0, salt la pasul 5”. Diferenţa este aceea că o instrucţiune de salt condiţionat efectuează saltul numai dacă este îndeplinită o anumită condiţie. De exemplu, sevcevenţa de instrucţiuni din figura 2.3 reprezintă un algoritm pentru împărţirea a două valori, în care pasul 3 descrie o instrucţiune de salt condiţionat care realizează protecţia împotriva împărţirii cu zero.

Figura 2.3 Împărţirea a două valori stocate în memorie

Pasul 1: Se încarcă (LOAD) un registru cu o valoare din memorie.
Pasul 2: Se încarcă (LOAD) alt registru cu altă valoare din memorie.
Pasul 3: Dacă a doua valoare este zero, salt (JUMP) la pasul 6.
Pasul 4: Se împarte conţinutul primului registru la conţinutul celui de-al doilea registru şi se depune rezultatul într-un al treilea registru.
Pasul 5: Se stochează (STORE) conţinutul celui de-al treilea registru în memorie.
Pasul 6: Stop.

2.2 Stocarea programelor

Primele calculatoare nu excelau în flexibilitate, deoarece programul executat de fiecare dispozitiv era cablat în unitatea de comandă ca o parte a sistemului. Un asemenea sistem este similar unei cutii muzicale care cântă întotdeauna aceeaţi melodie, când de fapt avem nevioe de posibilităţile complexe oferite de un cititor de CD. Una dintre soluţiile utilizate la primele calculatoare electronice pentru a dobândi mai multă flexibilitate a constituit-o proiectarea unităţilor de control astfel încât diversele blocuri să poată fi reconectate după nevoie. Acest lucru se realiza prin intermediul unei plăci de conexiuni realizate pe principiul vechilor plăci de comutare utilizate în centralele telefonice, la care capetele firelor de legătură erau fixate în găurile plăcii.

2.2.1 Instrucțiunile ca şiruri de biți

Un pas înainte (cu care a fost creditat, poate pe nedrept, John von Neumann - unii afirmă că această idee a aparținut de fapt lui J.P. Eckert Jr. de la Moore School, dar că ideea a devenit parte integrantă a unui efort de grup şi în final a fost atribuită în mod eronat lui von Neumann) s-a făcut o dată cu înțelegerea faptului că, în mod similar datelor, un program poate fi codificat şi stocat în memoria principală a calculatorului. Dacă unitatea de comandă este proiectată astfel încât să extragă programul din memorie, să decodifice instrucțiunile şi apoi să le execute, programul unui calculator poate fi schimbat pur şi simplu prin modificarea conținutului memorie, în loc să se reconecteze blocurile unității de comandă a calculatorului.

Conceptul de program stocat în memorie a devenit în prezent soluția standard de lucru. Pentru a-l putea aplica, calculatorul este proiectat astfel încât să recunoască anumite modele de biți ca reprezentând diferite instrucțiuni.Această colecție de instrucțiuni, împreună cu sistemul de codificare, poartă numele de **limbaj maşină (machine-language)** – deoarce defineşte modul în care comunicăm calculatorului algoritmii pe care trebuie să-i execute.

Versiunea codificată a unei instrucțiuni maşină constă de obicei din două părți: **câmpul codului de operație (operation code** – prescurtat op-code) şi **câmpul operandului (operand).** Şirul de biți care apare în câmpul opcodului specifică operația elementară, ce de exemplu STORE, SHIFT, XOR sau JUMP, a cărei execuție este solicitată de instrucțiune. Modelul de biți din câmpul operandului oferă informații detaliate asupra operației respective. De exemplu, în cazul unei operații de tip STORE, informația din câmpul operandului precizează registrul care conține datele care trebuie stocate, precum şi care este celula din memorie în care se vor stoca ele.

Conceptul stocării programului în memorie nu este deloc complicat. Ceea ce l-a făcut dificil de înțeles la început a fost faptul că toată lumea se gândea la programe şi date ca fiind lucruri complet diferite: datele se stochează în memorie, programele reprezintă părți componente ale unității de comandă. Este o ilustrare foarte bună a expresiei "a nu vedea pădurea din cauza copacilor". Este uşor să devii prizonierul unor asemenea tipare, şi nu putem şti dacă nu cumva dezvoltarea informaticii este şi astăzi încorsetată de asemenea limitări. De care nu ne dăm seama. Într-adevăr, o mare parte din frumusețea științei e dată de faptul că în permanență apar idei noi, care deschid uşi către noi teorii şi aplicații.

2.2.2 Un limbaj maşină tipic

Să vedem cum ar trebui codificate instrucțiunile unui calculator obişnuit. Calculatorul pe care-l vom utiliza în cadrul discuției noastre este descris în Anexa C şi prezentat în figura 2.4. El are şaisprezece regiştri de uz general, numerotați de la 0 la F în hexazecimal, iar memoria sa conține 256 celule. În consecință, fiecare celulă de memorie este desemnată individual, sau identificată, printr-un întreg între 0 şi 255. După cum am menționat anterior, de obicei se utilizează celule care au opt biți, aşa că vom considera că şi celulele de memorie ale calculatorului nostru au această mărime. Cum regiştrii de uz general sunt utilizați pentru stocarea temporară a datelor din memorie, vom considera că fiecare registru are de asemenea dimensiunea de opt biți.

Figura 2.4 Arhitectura calculatorului descris în Anexa C

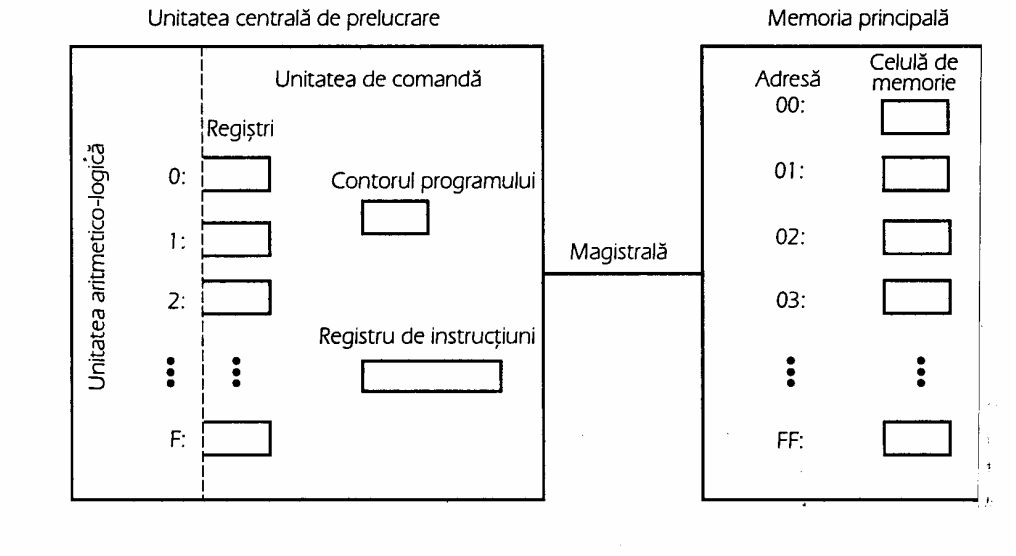
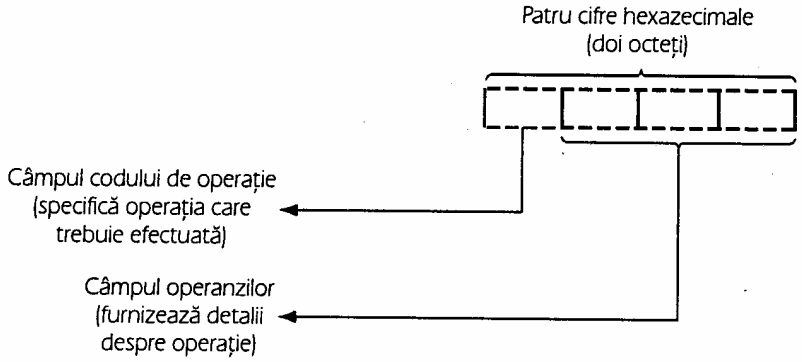


Figura 2.5 Formatul unei instrucţiuni maşină pentru calculatorul descris în Anexa C



➢ Coduri de operaţie

Studiind limbajul maşină prezentat în Anexa C, veţi constata că fiecare instrucţiune este codificată pe 16 biţi, reprezentaţi cu ajutorul a patru cifre hexazecimale (ca în figura 2.5). Codul de operaţie al fiecărei instrucţiuni este reprezentat de primii patru biţi, sau, ceea ce revine la acelaşi lucru, de prima cifră hexazecimală. Lista completă a instrucţiunilor conţine numai 12 instrucţiuni elementare, ale căror opcoduri sunt reprezentate prin cifrele hexazecimale de la 1 la C. Astfel, orice cod de instrucţiune care începe cu cifra hexazecimală 3 (şirul de biţi 0011) se referă la o instrucţiune de stocare (STORE), iar orice opcod care începe cu cifra hexazecimală A se referă la o instrucţiune ROTATE.

Calculatorul dispune de două instrucţiuni de adunare ADD: una pentru adunarea reprezentărilor în complement faţă de doi şi una pentru adunarea reprezentărilor în virgulă mobilă. Această tratare diferită este impusă de faptul că adunarea cuvintelor binare care reprezintă valori codificate cu notaţia în complement faţă de doi necesită executarea în interiorul unităţii aritmetico-logice a unor acţiuni diferite de cele de la adunarea valorilor reprezentate în virgulă mobilă.

➢ Operanzi

Să aruncăm o privire şi asupra câmpului opranzilor. El constă din trei cifre hexazecimale (12 biţi) şi în fiecare caz (exceptând instrucţiunea HALT, care nu necesită precizări suplimentare) clarifică instrucţiunea generală furnizată de câmpul codului de operaţie. De exemplu, dacă prima cifră hexazecimală a unei instrucţiuni este 1 (opcodul pentru încărcarea datelor din memorie), următoarea cifră hexazecimală a instrucţiunii va preciza în care registru trebuie încărcată valoarea, iar ultimele două cifre hexazecimale vor indica din care celulă de memorie este prelucrată aceasta. Astfel, instrucţiunea 1347 (hexazecimal) este codificată sub forma instrucţiunii "LOAD în registrul 3 conţinutul celulei de memorie de la adresa 47". În cazul codului de operaţie 7, care cere să se efectueze un SAU logic (OR) cu conţinutul a doi regişţri, a doua cifră hexazecimală specifică registrul în care trebuie depus rezultatul, iar ultimele două cifre hexazecimale indică regiştrii implicaţi în operaţia logică OR. Astfel, codul 70C5 se traduce prin instrucţiunea "Se execută OR cu conţinutul registrului C şi al registrului 5 şi se pune rezultatul în registrul 0".

Între cele două instrucţiuni LOAD ale calculatorului nostru există o diferenţă subtilă. Aşa cum se poate observa, codul de operaţie 1 (hexazecimal) se referă la instrucţiunea care încarcă într-un registru conţinutul unei celule de memorie, în timp ce codul de operaţie 2 (hexazecimal) se referă la instrucţiunea care încarcă într-un registru o anumită valoare particulară. Diferenţa constă în faptul că în primul caz câmpul operand conţine o adresă, în timp ce în cel de-al doilea caz el conţine valoarea binară care trebuie încărcată în registru.

O situaţie interesantă apare şi în cazul instrucţiunii JUMP (codul hexazecimal B). Prima cifră hexazecimală indică registrul care trebuie comparat cu registrul 0. Dacă acest registru are acelaşi conţinut ca şi registrul 0, se sare la instrucţiunea de la adresa indicată de ultimele cifre hexazecimale din câmpul operandului. Altfel, se continuă execuţia normală a programului. În general, în acest mod se realizează un salt condiţionat. Dar dacă prima cifră hexazecimală din câmpul operandului este 0, instrucţiunea cere ca registrul 0 să fie comparat cu el însuşi. De vreme ce un registru este întotdeauna egal cu el însuşi, saltul va fi întotdeauna efectuat. În consecinţă, orice instrucţiune al cărei cod începe cu cifrele hexazecimale B0 va fi interpretată ca un salt necondiţionat.

➢ Exemplu de program

Vom încheia acest subcapitol cu prezentarea versiunii codificate a instrucţiunilor redate în figura 2.2. Vom presupune că valorile care trebuie adunate sunt stocate în notaţia în complement faţă de doi la adresele de memorie 6C şi 6D, iar suma urmează a fi plasată în memorie la adresa 6E.

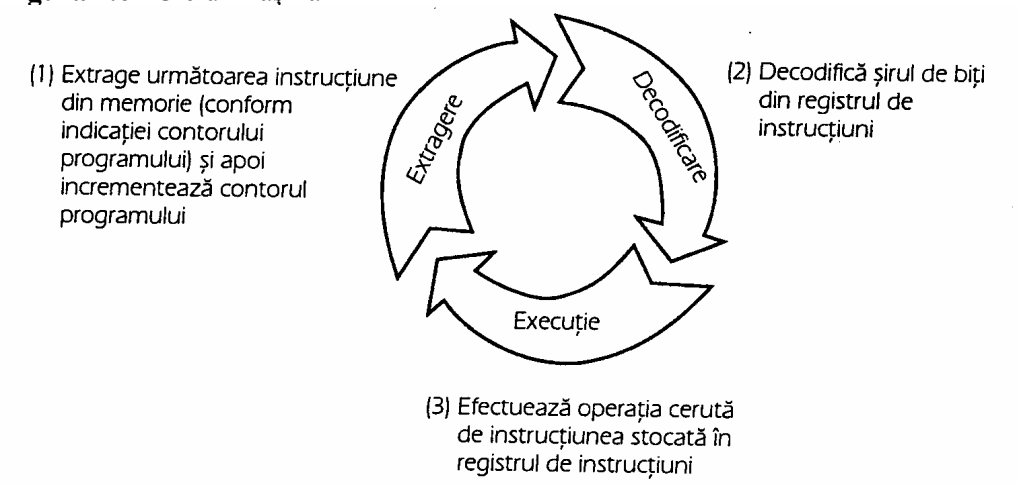
Pasul 1: 156C
Pasul 2: 166D
Pasul 3: 5056
Pasul 4: 306E
Pasul 5: C000

2.3 Execuția programelor

Calculatorul urmăreşte un program stocat în memorie copiind instrucțiunile din memorie în unitatea de comandă pe măsură ce are nevoie de ele. O dată ajunsă în unitatea de comandă, fiecare instrucțiune este decodificată şi executată. Ordinea în care sunt extrase instrucțiunile din memorie corespunde ordinii în care acestea sunt stocate, cu excepția cazului în care o instrucțiune JUMP specifică altceva. Pentru a înțelege cum se desfăşoară procesul de execuție a unui program trebuie să studiem mai amănunțit unitatea de comandă din interiorul CPU. Ea conține doi regiştri cu destinație specială: **contorul programului (program counter)** şi **registrul de instrucțiuni (instruction register)**, aşa cum se poate observa în figura 2.4. Registrul contorului programului conține adresa următoarei instrucțiuni care trebuie executată, permițând astfel calculatorului să urmărească locul în care se află în program. Registrul de instrucțiuni este utilizat pentru stocarea instrucțiunii în curs de execuție.

Unitatea de comandă îşi realizează sarcinile repetând continuu un algoritm, denumit **ciclul maşinii** (**machine cycle**), care constă din trei paşi: extragere (fetch), decodificare şi execuție (figura 2.6). În timpul pasului de extragere, unitatea de comandă solicită memoriei principale să-i furnizeze următoarea instrucțiune care va fi executată. Unitatea ştie unde se află următoarea instrucțiune în memorie deoarece adresa acesteia se află în registrul contorului programului. Unitatea de comandă plasează instrucțiunea recepționată din memorie în registrul său de instrucțiuni şi incrementează apoi contorul programului astfel încât acesta să conțină adresa următoarei instrucțiuni.

Figura 2.6 Ciclul maşină



Având acum instrucțiunea în registrul de instrucțiuni, unitatea de comandă începe faza de decodificare din ciclul maşinii. În acest punct, ea analizează câmpurile codului de operație şi operanzilor pentru a determina ce acțiuni trebuie efectuate.

După decodificarea instrucțiunii, unitatea de comandă intră în faza de execuție, în timpul căreia activează circuitele adecvate pentru realizarea acțiunilor solicitate. De exemplu, dacă instrucțiunea se referă la încărcarea datelor din memorie, unitatea de comandă face să se efectueze operația de încărcare; dacă instrucțiunea se referă la o operație aritmetico-logică, unitatea de comandă activează circuitele adecvate din unitatea aritmetico-logică, având ca intrări regiştrii corespunzători.

După execuția instrucțiunii unitatea de comandă începe un nou ciclu al maşinii cu faza de extragere. Observați că deoarece contorul programului a fost incrementat la sfârşitul fazei precedente de extragere, el furnizează din nou unității de comană adresa corectă a instrucțiunii.

Un caz ceva mai aparte este reprezentat de execuția unei instrucțiuni JUMP. Considerăm de exemplu instrucțiunea B258, care se traduce ca "JUMP la instrucțiunea de la adresa 58 dacă conținutul registrului 2 este identic cu cel al registrului 0". în acest caz, faza de execuție din ciclul maşină începe cu compararea regiştrilor 2 şi 0. Dacă cei doi regiştri diferă, faza de execuție se termină şi începe următoarea fază de extragere. Dacă însă conținutul celor doi regiştri este identic, unitatea de comandă plasează valoarea 58 în contorul programului înainte de a termina faza de execuție. În această situație, următoarea fază de extragere începe cu valoarea 58 în contorul programului, aşa că instrucțiunea de la acea adresă va deveni prima instrucțiune de executat.

➢ Exemplu de execuție a unui program

Să studiem ciclul maşinii utilizat pentru programul pe care l-am codificat la sfârşitul subcapitolului 2.2. Mai întâi trebuie să stocăm programul în memorie. În exemplul nostru, vom presupune că programul este stocat la adrese succesive de memorie, începând de la adresa A0 în hexazecimal. în figura 2.7 este prezentat un tabel care descrie conținutul acestei zone de memorie. Având programul astfel memorat, putem provoca execuția lui plasând adresa primei instrucțiuni (A0) în registrul contorului programului şi pornind calculatorul.

Figura 2.7 Stocarea în memorie a programului de "adunare", începând de la adresa A0

Adresă	Conținut
A0	15
A1	6C
A2	16
A3	6D
A4	50
A5	56
A6	30
A7	6E
A8	C0
A9	00

Unitatea de comandă începe faza de extragere încărcând instrucțiunea de la adresa A0 (156C) şi plasând-o în registrul său de instrucțiuni. Observați că instrucțiunile utilizate de calculatorul nostru au lungimea de 16 biți (doi octeți). De aceea, instrucțiunea care trebuie extrasă ocupă atât celula de memorie de la adresa A0, cât şi celula de memorie de la adresa A1. Unitatea de comandă este proiectată ținându-se cont de acest lucru, aşa că ea încarcă conținutul ambelor celule de memorie şi plasează datele în registrul de instrucțiuni, care are o lungime de 16 biți.

Apoi, unitatea de comandă creşte cu 2 valoarea din contorul programului, astfel încât acest registru conține acum adresa următoarei instrucțiuni. La sfârşitul fazei de extragere din primul ciclu al maşinii, registrul contor al programului şi registrul de instrucțiuni conțin următoarele:

Contorul programului: A2
Registrul de instrucțiuni: 156C

Apoi, unitatea de comandă analizează instrucțiunea din registrul său de la instrucțiuni şi trage concluzia că trebuie să încarce în registrul 5 conținutul celulei de memorie de la adresa 6C. Această activitate de încărcare este efectuată în timpul fazei de extragere din următorul ciclu al maşinii. În timpul acestei faze de extragere unitatea de comandă obține instrucțiunea 166D din cele două celule succesive de memorie de la adresa A2, plasează această instrucțiune în registrul său de instrucțiuni şi incrementează apoi contorul programului la valoarea A4. Astfel, valorile din registrul contor al programului şi din registrul de instrucțiuni devin:

Contorul programului: A4
Registrul de instrucțiuni: 166D

Unitatea de comandă decodifică instrucțiunea 166D şi află că trebuie să încarce în registrul 6 conținutul celulei de memorie de la adresa 6D. Se intră apoi în faza de execuție, în timpul căreia registrul 6 este încărcat cu valoarea cerută.

Cum contorul programului conține acum valoarea A4, unitatea de comandă extrage următoarea instrucțiune care începe la această adresă. Ca rezultat, în registrul de instrucțiuni este plasat codul 5056, iar contorul programului este incrementat la valoarea A6. Unitatea de comandă decodifică acum conținutul registrului de instrucțiuni şi intră în faza de execuție, activând circuitele de adunare în complement față de doi având ca intrări regiştrii 5 şi 6.

În timpul fazei de execuție, unitatea aritmetico-logică efectuează operația de adunare cerută, lasă rezultatul în registrul 0 (după cum îi cere unitatea de comandă) şi raportează unității de comandă terminarea operației. Apoi unitatea de comandă începe o altă fază de extracție dintr-un ciclu al maşinii. Încă o dată, cu ajutorul contorului programului, ea extrage următoarea instrucțiune (303E) din cele două celule succesive de memorie începând de la adresa A6 şi incrementează contorul programului la A8. Instrucțiunea este decodificată în timpul următoarei faze de decodificare şi executată în următoarea fază de execuție. în acest moment, rezultatul adunării este plasat în celula de memorie de la adresa 6E.

Este extrasă apoi următoarea instrucțiune, care începe de la adresa de memorie A8, şi contorul programului este incrementat la valoarea AA. Conținutul registrului de instrucțiuni (C000) este decodificat în timpul următoarei faze de execuție din ciclul maşinii şi programul se termină.

Pe scurt putem observa că execuția unui program stocat în memorie nu are nimic misterios. De fapt, procesul este similar cu cel pe care îl punem în practică atunci când trebuie să executăm o listă detaliată de instrucțiuni. În timp ce noi ştim unde ne aflam în listă marcând instrucțiunile pe măsură ce le execută, calculatorul memorează acest lucru cu ajutorul contorului programului. După ce determinăm următoarea instrucțiune pe care trebuie să o executăm, citim instrucțiunea si o interpretăm, la fel cum calculatorul decodifică instrucțiunile. În sfârşit , efectuăm operația cerută şi revenim la următoarea instrucțiune din listă, la fel ca un calculator care îşi execută instrucțiunile în faza de execuție şi continuă cu o noua fază de extragere.

Diferența esențială dintre oameni şi calculatoare în ceea ce priveşte execuția unor astfel de liste de instrucțiuni este dată de acuratețe si viteza de execuție. Unitatea de comandă repetă mereu ciclul maşinii, fără a încerca să o ia pe scurtătură şi, în consecință, fără erori. Pe de altă parte, oamenii se plictisesc repede, cred că au înțeles exact ce au de făcut şi se grăbesc să treacă la fapte, fără a acorda deplină atenție fiecărei instrucțiuni.

În ceea ce priveşte viteza, tehnologia continuă să ne uimească cu echipamente din ce în ce mai rapide. În prezent calculatoarele a căror viteză de lucru se măsoară în milioane de instrucțiuni pe secunda (MIPS) nu sunt ceva neobişnuit, iar domeniul uzual în care se situează vitezele de lucru este intre 10 MIPS si 100 MIPS. Se pare însă că există limite în ceea ce priveşte viteza cu care calculatoarele pot executa instrucțiunile, iar tehnologia se orientează deja către alte alternative.

➢ Programe şi date

Mai multe programe pot fi stocate simultan în memoria principală a unui calculator, atâta timp cât ocupă zone de memorie diferite, iar prin setarea adecvată a contorului programului se poate determina care program se va executa la pornirea calculatorului .

Trebuie însă să avem în vedere ca deoarece datele sunt stocate de asemenea în memorie şi sunt codificate tot cu cifre binare (0 si 1), calculatorul nu poate face distincția între date şi program. Dacă în registrul contorului de program este încărcată în loc de adresa programului o adresă unei zone de date, calculatorul, extrage codurile corespunzătoare datelor ca şi cum ar fi instrucțiuni şi le execută; rezultatul final depinde de datele respective.

Acest lucru nu este ceva rău. Încă o dată, ideea de a privi programele şi datele ca fiind entități complet diferite reprezintă un punct îngust de vedere, care trebuie depăşit. În realitate, existența unui aspect comun pentru programele şi datele stocate în memoria calculatorului s-a dovedit a fi o caracteristică utilă, pentru că permite unui program să manipuleze alte programe (sau chiar pe el însuşi) ca pe nişte date. Aşa cum vom vedea, ceea ce un program consideră a fi date se poate dovedi a fi de fapt un program.

ANEXA C

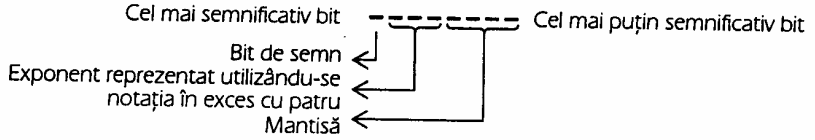
Exemplu de limbaj mașină tipic

Arhitectura calculatorului

Calculatorul are 16 regiștri de uz general, numerotați de la 0 la F (în hexazecimal). Fiecare registru are lungimea de un octet (opt biți). Pentru identificarea regiștrilor în cadrul instrucțiunilor, fiecare registru are atribuit un cuvânt binar unic pe patru biți care reprezintă numărul său. Astfel, registrul 0 este identificat prin 0000 (0 în hexazecimal), iar registrul 4 este identificat prin 0100 (4 în hexazecimal).

Memoria principală conține 256 de celule. Fiecare celulă de memorie conține opt biți (sau un octet) de date. Cum în memorie sunt 256 de celule, fiecare celulă are alocată o adresă unică ce constă dintr-un număr întreg în intervalul de la 0 la 255. În consecință, o adresă poate fi reprezentată printr-un șir de opt biți între 00000000 și 11111111 (sau printr-o valoare hexazecimală din intervalul de la 00 la FF).

Valorile în virgulă mobilă sunt stocate utilizându-se formatul prezentat în continuare.



Limbajul mașină

Fiecare instrucțiune mașină are o lungime de doi octeți. Primii patru biți alcătuiesc câmpul codului de operație; ultimii doisprezece biți formează câmpul operandului. Tabelul următor prezintă instrucțiunile mașină, în notație hexazecimală și însoțite de o scurtă descriere a fiecăreia. Literele R, S și T sunt utilizate în locul cifrelor hexazecimale pentru a reprezenta identificatori de regiștri. Literele X și Y sunt utilizate în locul cifrelor hexazecimale în diferite câmpuri care nu reprezintă regiștri.

Cod operație	*Operand*	*Descriere*
1	RXY	Încarcă (LOAD) registrul R cu valoarea găsită în celula de memorie a cărei adresă este XY. *Exemplu*: 14A3 va avea ca rezultat plasarea conținutului celulei de memorie cu adresa A3 în registrul 4.
2	RXY	Încarcă (LOAD) registrul R cu valoarea reprezentată de șirul de biți XY. *Exemplu*: 20A3 va avea ca rezultat înscrierea valorii A3 în registrul 0.
3	RXY	Stochează (STORE) valoarea din registrul R în celula de memorie a cărei adresă este XY. *Exemplu*: 35B1 va avea ca rezultat plasarea conținutului registrului 5 în celula de memorie cu adresa B1.
4	0RS	Mută (MOVE) conținutul registrului R în registrul S. *Exemplu*: 40A4 va avea ca rezultat copierea conținutului registrului A în registrul 4.
5	RST	Adună (ADD) șirurile de biți din regiștrii S și T ca și cum ar fi reprezentări în complement față de doi și depune rezultatul în registrul R. *Exemplu*: 5726 va avea ca rezultat adunarea valorilor din regiștrii 2 și 6 și plasarea rezultatului în registrul 7.
6	RST	Adună (ADD) șirurile de biți din regiștrii S și T ca și cum ar fi reprezentări în virgulă mobilă și depune rezultatul în registrul R. *Exemplu*: 634E va avea ca rezultat adunarea valorilor din regiștrii 4 și E ca valori reprezentate în virgulă mobilă și plasarea rezultatului în registrul 3.
7	RST	Execută un sau logic (OR) între șirurile de biți din regiștrii S și T și depune rezultatul în registrul R. *Exemplu*: 7CB4 va avea ca rezultat executarea unui sau logic între conținuturile regiștrilor B și 4 și plasarea rezultatului în registrul C.
8	RST	Execută un și logic (AND) între șirurile de biți din regiștrii S și T și depune rezultatul în registrul R. *Exemplu*: 8045 va avea ca rezultat executarea unui și logic între conținuturile regiștrilor 4 și 5 și plasarea rezultatului în registrul 0.
9	RST	Execută un sau exclusiv (XOR) între șirurile de biți din regiștrii S și T și depune rezultatul în registrul R. *Exemplu*: 95F3 va avea ca rezultat executarea unui sau exclusiv între conținuturile regiștrilor F și 3 și plasarea rezultatului în registrul 5.
A	R0X	Rotește (ROTATE) șirul de biți din registrul R cu un bit la dreapta de X ori. La fiecare rotație, bitul cel mai puțin semnificativ este mutat pe poziția bitului cel mai semnificativ. *Exemplu*: A403 va avea ca rezultat mutarea circulară la dreapta a conținutului registrului 4 de 3 ori.
B	RXY	Salt (JUMP) la instrucțiunea plasată în celula de memorie cu adresa XY dacă conținutul registrului R este egal cu conținutul registrului 0. Altfel, se continuă execuția normală a secvenței de instrucțiuni. *Exemplu*: B43C va compara mai întâi conținutul registrului 4 cu conținutul registrului 0. Dacă cei doi regiștri sunt identici, secvența de execuție va fi modificată astfel încât următoarea instrucțiune care se va executa să fie cea aflată la adresa de memorie 3C. Altfel, execuția programului va continua în mod normal.
C	000	Oprirea (HALT) execuției. *Exemplu*: C000 va avea ca rezultat oprirea execuției programului.